

# Créez une plateforme pour amateurs de Nutella

---

## 1 - Planification du projet

[Dépot Github](#)

[Tableau Trello](#)

[Analyse fonctionnelle](#)

[Application sur Heroku](#)

## 2 - Création d'un nouveau projet Django

création d'un environnement virtuel avec pipenv puis installation du packet django 2.2

Création du projet

```
django-admin startproject purbeurre
cd purbeurre
manage.py startapp
```

création de variables d'environnement

`.env`

```
ENV=DEV
SECRET_KEY= [SECRETKEY]
DB_NAME=purbeurre
DB_USER=root
DB_PASSWORD=root
```

Modification du fichier de configuration

- Langue et fuseau horaire
- Configuration PROD/DEV
- Ajout de la fonction `get_env_variable()`
- Configuration de la base de donnée

Création des dossiers static et templates

```
└─ purbeurre
   └─ authentication.....=> Authentication application
      └─ __init__.py
      └─ admin.py
      └─ apps.py
```

```
| | | forms.py
| | | migrations
| | | models.py
| | | static
| | | templates
| | | | authentication
| | | | | account.html
| | | | | register.html
| | | | registration
| | | | | login.html
| | | tests
| | | | __init__.py
| | | | test_account.py
| | | | test_login.py
| | | | test_logout.py
| | | | test_register.py
| | | urls.py
| | | views.py
| | products.....=> Products application
| | | __init__.py
| | | admin.py
| | | apps.py
| | | management
| | | | commands
| | | | | fillindb.py
| | | migrations
| | | models.py
| | | templates
| | | | products
| | | | | base_product_title_img.html
| | | | | detail.html
| | | | | myproducts.html
| | | | | result.html
| | | | | search.html
| | | | | substitute_confirm_delete.html
| | | tests
| | | | __init__.py
| | | | pytest_fixture.py
| | | | test_delete.py
| | | | test_detail.py
| | | | test_myproducts.py
| | | | test_result.py
| | | | test_save.py
| | | | test_search.py
| | | urls.py
| | | views.py
| | purbeurre.....=> Project folder
| | | __init__.py
| | | settings.py
| | | static
| | | tests
| | | | __init__.py
| | | | | test_purbeurre.py
| | test_settings.py
```

```
|   ├── urls.py
|   └── wsgi.py
├── manage.py
├── pytest.ini
├── static .....=> core statics
└── templates.....=> core templates
```

---

## 3 - Mise en place du Front

Réalisation de 5 pages à partir du template pour correspondre au cahier des charges. (doc/front/\*)

- Accueil
- Résultats / Mes aliments
- Page Aliment
- Mon Compte
- Mention légales

Découpage avec une partie base et différents blocs.

---

## 4 - Purbeurre

C'est le module principal du projet.

il y a les composants communs dans les dossiers *template* et *static*.

index : la page d'accueil du site

J'ai utilisé la vue générique `TemplateView` directement dans `urlpatterns` en définissant le *template* cible.

```
urlpatterns = [
    path('', TemplateView.as_view(template_name='products/index.html'),
    name='index'),
]
```

J'ai placé ici et procédé de même pour la page de mentions légales.

Il y a également l'import des *ulrpattern* des application sont préfixées.

---

Compétences acquises sur ce module.

- Utilisation de vue générique directement dans `urlpatterns` (*index*)

## 5 - Authentification

Ce module permet d'adapter le module *User* intégré à Django pour l'utiliser avec l'adresse email à la place du *username*

[purbeurre/authentication/models.py](#)

Surcharge du model *User* afin de l'utiliser avec l'adresse email. Il a fallut créer un *Manager* pour ce changement: `MyUserManager`

---

login : permet à l'utilisateur de se connecter

```
urlpatterns = [  
    path('login/', auth_views.LoginView.as_view(), name='login'),  
]
```

J'ai utilisé la vue générique `LoginView` qui appelle le template `registration/login.html` par défaut.

---

logout : déconnecte l'utilisateur.

J'ai utilisé le décorateur `@login_required` qui permet de renvoyer vers l'adresse définie dans `settings.LOGIN_URL` si l'utilisateur n'est pas authentifié.

J'ai utilisé le raccourci `redirect`.

```
@login_required  
def LogoutView(request):  
    logout(request)  
    return redirect('products:index')
```

---

register : création d'un utilisateur.

J'ai utilisé la vue générique `FormView` et le formulaire générique `UserCreationForm` qu'il a fallut adapter pour prendre en compte l'*email* à la place de *username*.

Utilisation de la méthode `form_valid` pour enregistrer le formulaire

---

account : affichage de la page d'information sur l'utilisateur.

J'ai utilisé le Mixin `LoginRequiredMixin` pour vérifier que l'utilisateur est bien connecté.

Utilisation de la méthode `get` pour afficher la page et récupérer les informations sur l'utilisateur (en session dans `request`).

---

Compétences acquises sur ce module.

- Surcharge de *model* de base de Django (*User*)
- Utilisation de vues génériques (*LoginView*, *FormView*)
- Utilisation de formulaire générique et adaptation (*UserCreationForm*)

- Utilisation de la vérification d'authentification de 2 manières (décorateur pour les "vues fonctions" et Mixin pour les "vues class")
  - Utilisation de raccourcis (*render*, *redirect*)
  - Utilisation de CBV (*AccountView*, *RegisterView*)
- 

## 6 - Products

Mise en place d'une commande personnalisée `django-admin` pour peupler la base de donnée

```
└─ products
  └─ management
    └─ commands
      └─ fillindb.py
```

La commande:

```
python manage.py fillindb 50
```

---

search : recherche d'un produit dans la base.

J'ai utilisé une vue générique `ListView`.

- Je récupère l'élément à rechercher *query* qui est une requete `GET`
- Je définie la requete dans la base de donnée avec la méthode `get_queryset`
- Je rajoute des éléments à `context` pour mon template avec la methode `get_context_data`

[products/search.html](#)

Je fais une boucle sur `object_list` pour afficher les produits.

Il y a un test sur `object_list` pour afficher un message si la requete ne renvoie rien.

Mise en place de la pagination pour afficher des pages de 9 produits.

---

result : recherche de produits de substitution dans la base.

J'ai utilisé une vue générique `ListView`.

Requête sur la liste de produit:

```
Product.objects\
    .filter(category=self.product.category)\
    .filter(nutrition_grades__lte=self.product.nutrition_grades)\
    .exclude(id=self.kwargs['product_id'])\
    .order_by('nutrition_grades')[:9]
```

1. Filtre sur la même catégorie que le produit d'origine
2. Filtre sur un *nutrition\_grades* supérieur ou égal
3. Exclusion du produit d'origine
4. Classement par *nutrition\_grades* (ordre croissant) et limitation aux 9 premiers produits

Récupération de la variable dans l'url

url.py

```
path('<int:product_id>/result/', views.ResultView.as_view(),  
name='result'),
```

views.py

```
self.product = Product.objects.get(pk=self.kwargs['product_id'])
```

Affichage d'un message si le produit à déjà été enregistré

1. récupération de *allreadysaved* dans `self.request.GET`
2. ajout dans `context`
3. test dans `product/result.html` pour afficher le message

---

detail : affiche les détail d'un produit.

J'ai utilisé une vue générique `DetailView`.

La requête se fait sur la clé primaire qui arrive dans l'url : `<int:pk>`

```
urlpatterns = [  
    path('<int:pk>/detail/', views.DetailProductView.as_view(),  
        name='detail'),  
]
```

Je récupère le nom du produit dans `kwargs['object']`

---

myproducts : affiche les pairs de produits enregistrés.

J'ai utilisé une vue générique `ListView`.

Il y a une requête sur la table `Substitute` filtré sur l'id de l'utilisateur

```
def get_queryset(self):  
    return  
    Substitute.objects.filter(user_id=self.request.user.id).order_by('-id')
```

Les données sont ensuite récupérées dans la page [products/myproducts.html](#)

- `object_list[0].product_id` : donne accès à la table Product pour le produit d'origine
- `object_list[0].substitute_id` : donne accès à la table Product pour le produit de substitution

Utilisation de `LoginRequiredMixin` pour protéger la page

Affichage d'un message si il n'y a pas de retour à la requête.

Mise en place de la pagination pour afficher des pages de 5 substituts.

---

`delete` : supprimer une paire de produit enregistré

J'ai utilisé une vue générique `DeleteView`

J'ai appelé la vue avec une requête `GET` afin de pouvoir afficher une page de confirmation.

le nom du template par défaut : `products/substitute_confirm_delete.html`

---

Compétences acquises sur ce module.

- Utilisation de vue générique (`ListView`, `DetailView`, `DeleteView`)
  - Définition d'une requête spécifique dans une vue générique (`get_queryset`)
  - Ajout d'éléments à `context` dans une vue générique (`get_context_data`)
  - Requête "complexe" (result)
  - Utilisation de l'utilisateur en session pour faire une requête (`myproducts`)
  - Récupération de données sur une table `ManyToMany`
  - Utilisation du nom du template générique (`delete`)
  - Utilisation du module `Paginator`
- 

## Partie Admin

Ajout de la table User dans la partie *admin*

- Possibilité de modifier nom/prénom
  - *email* est en lecture seule
  - Ajout de la somme des substituts enregistrés par utilisateur
- 

## 7 - Mise en ligne

Mise en production

Création de l'application sur l'interface web de Heroku puis ajout du remote

```
heroku git:remote -a bc-ocdapythonpr8
```

Activation de Postgres

```
heroku addons:create heroku-postgresql:hobby-dev
```

Définition de DATABASES

```
DATABASES = {  
    'default':  
        dj_database_url.config(conn_max_age=600, ssl_require=True)  
}
```

Création du [Procfile](#)

Création des variables d'environnement sur Heroku

```
heroku config:set ENV=PRODUCTION  
heroku config:set SECRET_KEY=['%secret_key%']
```

Envoi vers Heroku

```
git push heroku master
```

Migration et peuplage de la base

```
heroku run bash  
python purbeurre/manage.py migrate  
python purbeurre/manage.py fillindb 50
```

Mise en place de Travis : [https://travis-ci.org/Zepmanbc/oc\\_dapython\\_pr8](https://travis-ci.org/Zepmanbc/oc_dapython_pr8)

[.travis.yml](#)

Mise en place de Coveralls : [https://coveralls.io/github/Zepmanbc/oc\\_dapython\\_pr8](https://coveralls.io/github/Zepmanbc/oc_dapython_pr8)